Programming Fundamentals
Lecture #14
Pointers

Junaid Hassan

Lecturer CS & IT Department UOS MBDIN

junaidte14@gmail.com

What are Pointers?

- Pointers are variables whose values are memory addresses
- Normally, a variable directly contains a specific value. A pointer, on the other hand, contains an address of a variable that contains a specific value.
- In this sense, a variable name directly references a value, and a pointer indirectly references a value

How to define pointers in C Programming?

- int *countPtr, count;
- “countPtr is a pointer to int” or “countPtr points to an object of type int.”
- count = 5;
- countPtr = &count;

Pointer Operators:

& operator (will return pointer’s address)

* operator (will return pointer’s value)

Passing Arguments to Functions by Reference:

We already discussed this topic in array's chapter (ch #6)

Call by Value vs Call by Reference

Using the const Qualifier with Pointers:

The const qualifier enables you to inform the compiler that the value of a particular variable should not be modified

We discussed earlier that when we pass an array to a function then that call is a call by reference

- Which means that the modifications in the array elements will also modify the values of the original array elements
- If we don't want to modify the values of the original array then we use the const qualifier with array passed to the function

```
int funcCallByReference(const int *a){

//some code here

}
```

Pointer to Constant int: In above function if we tried to modify/change the value of variable a, then compiler will throw an error/warning that we cannot modify the value of that pointer.

```
int funcCallByReference(int * const a){

//some code here

}
```

Constant Pointer: In above function if we tried to modify/change the memory address of pointer a, then compiler will throw an error/warning that we cannot modify the memory address of that pointer.

Bubble Sort Using Call By Reference (Pointers)

```c
#include <stdio.h>
#define SIZE 10
void bubbleSort(int * const array, const int size);
void swap(int *element1Ptr, int *element2Ptr);
int main(void){

    int arr[SIZE] = {2, 3, 9, 4, 1, 5, 7, 12, 18, 15};

    printf("Original array values\n\n");
    for(int i=0; i<SIZE; i++){
        printf("%d\n", arr[i]);
    }

    printf("\n\nSorted array values\n\n");
    for(int i=0; i<SIZE; i++){
        printf("%d\n", arr[i]);
    }

}

void bubbleSort(int * const array, const int size){
    int pass, i=0;

    //array = &i; will generate error because we cannot modify the value of constant pointer named as array

    for(pass=0; pass < size-1; pass++){

        for(i=0; i < size-1; i++){

            if(array[i] > array[i+1]){
                swap(&array[i], &array[i+1]);
            }
        }
    }
}
```

```
void bubbleSort(int * const array, const int size){
    int pass, i=0;

    //array = &i; will generate error because we cannot modify the value of constant pointer named as array

    for(pass=0; pass < size-1; pass++){

        for(i=0; i < size-1; i++){

            if(array[i] > array[i+1]){
                swap(&array[i], &array[i+1]);
            }
        }
    }
}

void swap(int *element1Ptr, int *element2Ptr){
    int hold;
    hold = *element1Ptr;
    *element1Ptr = *element2Ptr;
    *element2Ptr = hold;
}
```

Sizeof Operator:

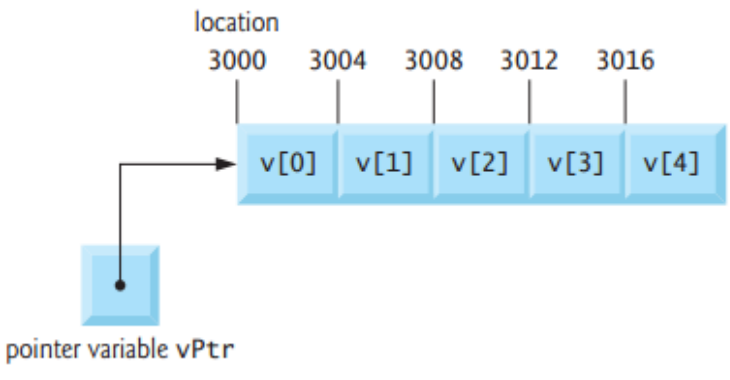
C provides the special unary operator sizeof to determine the size in bytes of an array (or
any other data type) during program compilation. When applied to the name of an array, the sizeof operator returns the total number of bytes in the array as an integer.

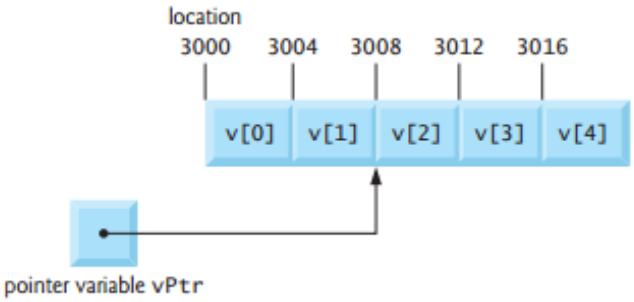
```c
/* Fig. 7.16: fig07_16.c
   Applying sizeof to an array name returns
   the number of bytes in the array. */
#include <stdio.h>

size_t getSize( float *ptr ); /* prototype */

int main( void )
{
   float array[ 20 ]; /* create array */

   printf( "The number of bytes in the array is %d"
           "\nThe number of bytes returned by getSize is %d\n",
           sizeof( array ), getSize( array ) );
   return 0; /* indicates successful termination */
} /* end main */

/* return size of ptr */
size_t getSize( float *ptr )
{
   return sizeof( ptr );
} /* end function getSize */
```

```
The number of bytes in the array is 80
The number of bytes returned by getSize is 4
```

Pointer Expressions and Arithmetic:

A limited set of arithmetic operations may be performed on pointers. A pointer may be incremented (++) or decremented (–), an integer may be added to a pointer (+ or +=),
an integer may be subtracted from a pointer (– or -=) and one pointer may be subtracted
from another





vPtr = &v[0]; OR vPtr = v; OR vPtr = &v;

vPtr += 2;

3008 i.e (3000 + 2*4)